UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

WILLER CARNEIRO QUINTANILHA

SIMPLENLP, UMA BIBLIOTECA PARA PROCESSAMENTO DE LINGUAGEM NATURAL

BAURU
Novembro/2019

WILLER CARNEIRO QUINTANILHA

SIMPLENLP, UMA BIBLIOTECA PARA PROCESSAMENTO DE LINGUAGEM NATURAL

Trabalho de Conclusão de Curso do Curso de Bacharel em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.
Orientador: Profª. Dra. Simone das Graças Domingues Prado

BAURU
Novembro/2019

Willer Carneiro Quintanilha SimpleNLP, uma biblioteca para processamento de linguagem natural/ Willer Carneiro Quintanilha. – Bauru, Novembro/2019-
37 p. : il. (algumas color.) ; 30 cm.
Orientador: Profª. Dra. Simone das Graças Domingues Prado
Trabalho de Conclusão de Curso – Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Ciência da Computação, Novembro/2019.
1. Processamento de Linguagem Natural 2. Linguagem 3. LSI 4. Clustering 5. Binary Encoding 6. Word Embbending

Willer Carneiro Quintanilha

SimpleNLP, uma biblioteca para processamento de linguagem natural

Trabalho de Conclusão de Curso do Curso de Bacharel em Ciência da Computação da Universidade Estadual Paulista "Júlio de Mesquita Filho", Faculdade de Ciências, Campus Bauru.

Banca Examinadora

Profª. Dra. Simone das Graças Domingues Prado
Orientador
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profª. Drª. Andréa Carla Gonçalves Vianna
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Profº. Drº. Kleber Rocha de Oliveira
Universidade Estadual Paulista "Júlio de Mesquita Filho"
Faculdade de Ciências
Departamento de Ciência da Computação

Bauru, _____ de _____ de _____.

Dedico este este trabalho a minha mãe que me apoiou durante todo este percurso sempre me apoiando e acreditando no meu futuro.

Agradecimentos

Agradeço a minha família, ao pessoal da república Alcatraz e ao professores do curso de Ciência da Computação da UNESP campus Bauru.

"If you hit the wall, you should thank God for placing it in front of you — that's when something new is born."

Akira Yoshino

Resumo

A linguagem é tida como a capacidade inata ao ser humano de se comunicar através de uma língua, a linguística é a ciência que ter por objetivo o estudo da linguagem humana, para a ligústica a semântica é o estudo do conteúdo das palavras, sentenças e enunciados, já a sintaxe investiga e determina os relacionamentos formais entre os elementos de uma linguagem.

A área de processamento de linguagem natural busca descrever técnicas para análise computacional de conjuntos de dados linguísticos.

Este trabalho visou a implementação de métodos para análise de dados linguísticos expressos na forma textual em uma biblioteca chamada SimpleNLP.

Palavras-chave: Processamento de Linguagem Natural, Latent Semantic Indexing, Linguagem, Semântica.

Lista de figuras

Lista de abreviaturas e siglas

PLN	Processamento de Linguagem Natural
LSI	Latent Semantinc Indexing

Sumário

1 Introdução

O objetivo da ciência linguística é caracterizar e explicar os fenômenos linguísticos que nos envolvem, em conversações, escritas e outros formatos de mídia. Parte deste processo tem haver com o lado cognitivo de como os humanos adquirem, produzem e entendem a linguagem, além do entendimento do relacionamento entre as expressões linguísticas e o mundo ("part of it has to do with understanding the relationship between linguistic utterances and the world") (MANNING; MANNING; SCHÜTZE, 1999).

Segundo Sapir todas as gramáticas vazam, a forma e o relacionamento do termos de uma língua são mutáveis, "This is because people are always stretching and bending the 'rules' to meet their communicative needs" (MANNING; MANNING; SCHÜTZE, 1999). Já para a questão do sentido, temos varias teorias diferentes que propõem conceitos para interpretação dos conteúdos de expressões, enunciados e termos.

A areá de processamento de linguagem natural busca descrever computacionalmente métodos capazes de analisar conjuntos de dados linguísticos.

O objetivo deste trabalho é o desenvolvimento de uma biblioteca para processamento de linguagem natural com o nome de SimpleNLP, os objetivos específicos a serem alcançados estão descritos a seguir:

a) Revisão bibliográfica sobre linguagem

b) Revisão bibliográfica sobre processamento de linguagem natural

c) Implementação de técnicas para processamento de linguagem natural em uma biblioteca de código livre chamada SimpleNLP

A metodologia de desenvolvimento deste trabalho buscou em um primeiro momento organizar o conhecimento linguístico e de processamento de linguagem natural, a fim de obter entendimento linguístico e computacional suficientes para desenvolvimento e fundamentação teórica de uma biblioteca para processamento de linguagem natural e aplicação prática desta, os passos realizados foram:

a) Pesquisa e organização de conhecimento a respeito da linguagem.

b) Pesquisa e organização de conhecimento a respeito de processamento de linguagem natural

c) Implementação de técnicas para processamento de linguagem natural em uma biblioteca de código livre chamada SimpleNLP

d) Aplicação das técnicas desenvolvidas

2 Linguagem

A linguagem é tida como a qualidade inata ao ser humano de se comunicar através de uma língua, "Convencionou-se atribuir o termo linguagem à capacidade geral que temos, enquanto seres humanos, de utilizar sinais com vistas à comunicação." (LEITE, 2010) todo ser humano não obstante seus atributos físicos ou de ser acometido por uma patologia é capaz de se comunicar através de um meio concreto ao qual chamamos língua, esta por sua vez evolui junto a cultura para satisfazer as necessidades comunicativas das sociedades.

A sintaxe é a "Parte da linguística que se dedica ao estudo das regras e dos princípios que regem a organização dos constituintes das frases" (PRIBERAM. . . , 2019), enquanto a semântica trata dos significados ou seja dos conteúdos expressos pela linguagem, na seção a seguir estão descritas algumas das teorias que buscam descrever o aspecto semântico da linguagem.

2.1 Teorias filosóficas semânticas da linguagem

As teorias semânticas filosóficas da linguagem buscam generalizar o processo de aquisição de sentido de forma que seja possível estabelecer princípios para a interpretação de qualquer enunciado, ou seja buscam explicar o componente semântico da linguagem.

> "Os filósofos dedicam muito tempo a esta questão: em que consiste o significado de uma expressão linguística? Esta é uma indagação filosófica. E, como acontece comumente em Filosofia, não se tem única resposta, mas várias. O problema com o qual as pesquisas, em IA e PLN, se deparam é o seguinte: qual das respostas que a Filosofia nos empresta torna possível, aos sistemas de computador, endereçar as necessidades da linguagem? E, se existe mais de uma, qual é a mais adequada?" (PINHEIRO, 2010)

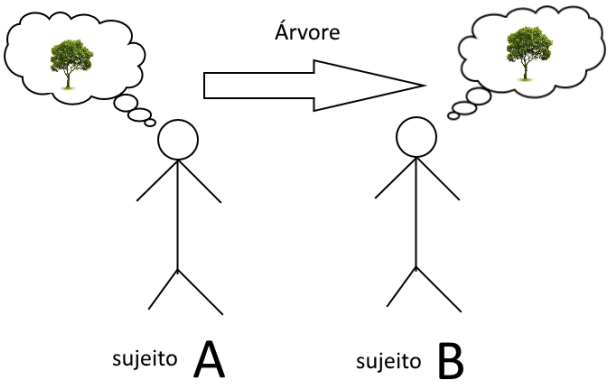
As próximas seções farão uma breve introdução as principais teorias semânticas filosóficas da linguagem

2.1.1 Teorias Ideacionais

As teorias semânticas ideacionais traziam a visão de que uma sentença corresponde a um estado mental. John Lock é o filósofo mais notório desta linha de teorias que considera que "os significados de expressões linguísticas são as ideias da mente" (PINHEIRO, 2010). tem se então que durante o processo comunicativo quem diz uma expressão tem uma ideia do que esta sendo dito e quem recebe esta expressão forma uma ideia do que foi dito, logo pode-se

dizer que o processo comunicativo ocorre de maneira satisfatória, quando os interlocutores formulam a mesma ideia do que foi dito assim como na Figura 1.

Figura 1 – Semântica para os teóricos ideacionais



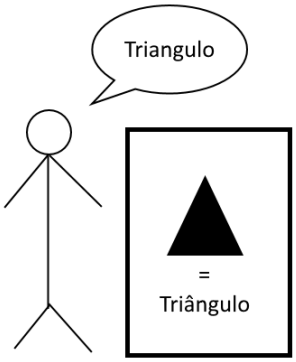
Fonte: Elaborada pelo autor.

Contudo essas teorias foram entrando em desuso no final do século XIX, principalmente em virtude de sua subjetividade e individualização do significado.

2.1.2 Teorias Representacionalistas

Das teorias a respeito dos significados na linguagem as mais facilmente aceitas pelo senso comum são as representacionalistas, estas tem "o entendimento de que a linguagem tem como principal função a de representar a realidade" (TEIXEIRA; MARTINS, 2008), logo significa o que representa, ver a figura: Figura 2.

Figura 2 – Semântica para os teóricos representacionalistas



Fonte: Elaborada pelo autor.

O quadro que se formara no século XX era o de que tem-se a realidade e o meio que usa-se para representa - la, este meio consiste em utilizar signos, para denotar, nomear e

designar as coisas do mundo, as expressões servem para descrever a forma como estas coisas estão posicionadas no mundo, como estas coisas se relacionam, e qual o estado destas coisas.

Para estes teóricos as palavras são mais significativas a medidas em que descrevem melhor coisas da realidade e as palavras sem valor representativo assumem funções auxiliares servindo apenas ao proposito da construção de representações mais complexas.

Esta teorias de abandonam a subjetividade dos significados, e tem fundamental simplicidade e capacidade de generalização.

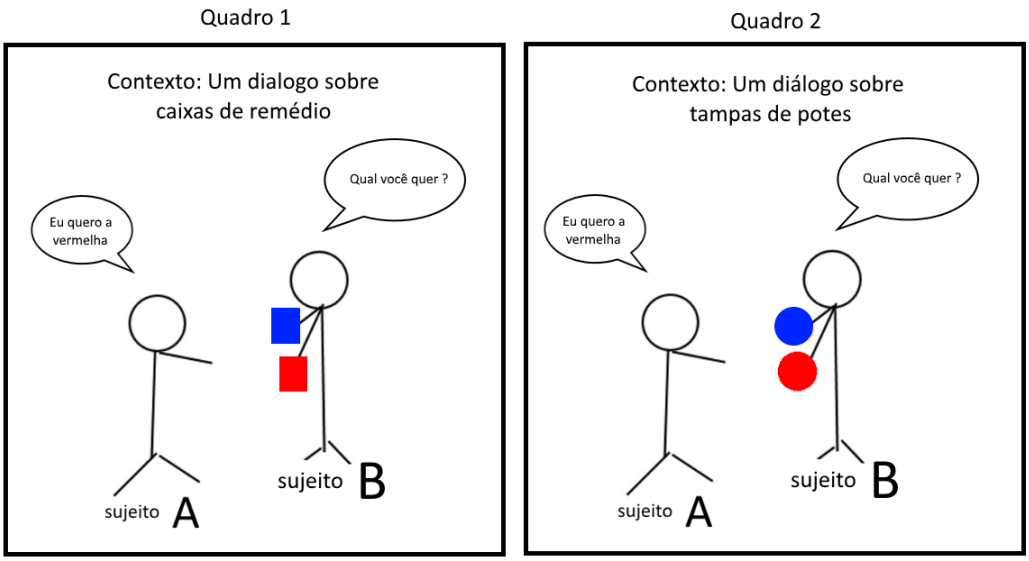
2.1.3 Teorias Pragmáticas

As teorias pragmáticas surgem como alternativa as teorias representacionalistas em meados do século XX tendo como principal autor Wittgenstein.

> "a 'lingüística' humboldtiana inaugura a investigação da linguagem ordinária, enquanto uma forma de ação e não de descrição do real, levando em consideração, sobretudo, o uso que fazemos da linguagem nos diferentes contextos." (NIGRO, 2009).

Para os teóricos desta linha de pensamento uma teoria de significado deveria descreve - lo em termos de uso da linguagem. A Figura 3 mostra como o uso também dito contexto impacta na interpretação das sentenças.

Figura 3 – O contexto para interpretação dos enunciados.



Fonte: Elaborada pelo autor.

Tem-se então uma teoria funcionalista de linguagem, que diz respeito a importância do uso ou contexto sobre a significação literal do mesmo.

Sendo assim para estes teóricos o significado de uma expressão não é determinado apenas pela expressão em si, mas também pelo contexto onde se encontra, ou seja a função da expressão é dada pelo contexto.

2.1.4 Teorias Inferencialistas

Estas teorias surgiram ao final do seculo XX, segundo os teóricos desta linha de pensamento as inferências prevalecem sobre a representação na ordem do processo de descrição do sentido em conceitos e sentenças.

> "Os termos de uma língua representam objetos do mundo real não pela propriedade primitiva de serem, respectivamente, representações e representados, mas em razão das participações dos conceitos (expressos pelos termos) em raciocínios, como premissas ou conclusões de inferências." (PINHEIRO, 2010)

Para estes teóricos entender o mundo significa compreender os relacionamentos expressos entre conceitos em raciocínios. Sellars mostra que o relacionamento inferencial denota a diferença entre relatos dados por humanos e maquinas. Enquanto humanos conseguem justificar as razões pelas quais deram determinados relatos as máquinas não conseguem. Pois as razões são expressas na forma de conhecimento inferencial, uma maquina pode dizer: "O céu é azul"mas desconhece as potenciais inferências derivadas do uso destes conceitos, como "azul"é uma cor, "azul"não é branco (Figura 4). Assim pode-se distinguir o nível de entendimento dos conceitos entre humanos e uma máquina.

Figura 4 – Uma abstração das teorias inferencialistas

Fonte: Elaborada pelo autor.

3 Processamento de Linguagem Natural

O Processamento de Linguagem Natural tem como objetivo obter informações de um conjunto de dados linguísticos, sejam estes representados em texto, áudio ou qualquer outro meio. Um dos grandes, se não, o maior desafio desta areá cabe a subjetividade da linguagem, dada que está sempre esta sujeita a interpretação do individuo, não podendo ser avaliada portanto como algo independente das experiências do interlocutores.

Contudo grandes massas de dados nos proporcionam a possibilidade de extrapolação da subjetividade da interpretação linguística única de cada individuo para a informação contida nas interpretações comuns a muitos indivíduos a exemplo do substantivo: "bolsa", os significados deste substantivo para cada individuo são diferentes, contudo estes significados se avaliados dentro do grupo de WhatsApp das pessoas que discutem a respeito do mercado de ações, tendem a significar o local onde se transacionam valores mobiliários ou seja a bolsa de valores.

Para a realização deste tipo de analise o processamento de linguagem natural fornece ferramental para representação e análise de dados linguísticos nos mais variados formatos.

As próximas subseções tratam de algumas das técnicas de PLN utilizadas para o processamento de texto.

3.1 Representações de dados textuais

Serão apresentadas a seguir formas de representação de dados textuais apropriadas a aplicação de técnicas algébricas, geométricas e de heurísticas e amplamente usadas no âmbito do Processamento de Linguagem Natural.

3.1.1 Tokenização

O processo de tokenização tange ao aspecto léxico e sintático da linguagem e busca a representação de texto na forma de *tokens*, geralmente os *tokens* são tidos como a menor unidade de sentido em um texto, habitualmente são desconsiderados para o processo de tokenização pontuações e espaços.

Considerando a menor unidade de sentido para um documento como sendo as palavras contidas no mesmo, "The entity word is one kind of token for NLP, the most basic one" (WEBSTER; KIT, 1992), pode-se definir o vetor das palavras de um documento como sua representação vetorial tokenizada (Figura 5).

Em suma o processo de tokenização é a representação de documentos na forma de conjuntos de *tokens*, sendo a forma de representação mais usada a de vetores de *tokens*.

Figura 5 – Um exemplo de tokenização

Fonte: Elaborada pelo autor

3.1.2 Binary Encoding

O processo de *Binary Encoding* permite a representação de documentos no formato de vetores numéricos, neste algoritmo primeiramente são extraídos e armazenados vetorialmente os *tokens* distintos presentes nos documentos a serem codificados, o vetor resultante desta operação serve como referência para a codificação e decodificação dos documentos, deve-se observar que este algoritmo não mantém a ordem dos dados codificados.

A seguir na Figura 6 são dados dois documentos de entada V e W

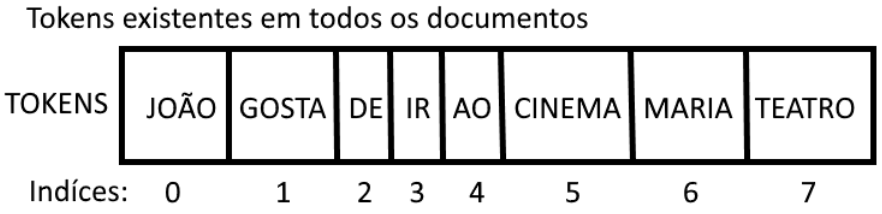
Figura 6 – Entradas V e W para o algoritmo de *Binary Encoding*

Fonte: Elaborada pelo autor

A primeira etapa para a codificação dos documentos segundo a técnica de *Binary Encoding* é a extração dos *tokens* presentes nos documentos a serem processados e armazenamento destes em um vetor, assim como na Figura 7, onde foram extraídos e armazenados no vetor $TOKENS$ os *tokens* presentes nos documentos V e W apresentados na Figura 6, este vetor

resultante atribui uma representação numérica a cada *token* presente nos documentos, esta representação é dada pelo índice de cada *token* no vetor resultante.

Figura 7 – Extração dos *tokens* de V e W *Binary Encoding*



Fonte: Elaborada pelo autor

É utilizada como base para representação dos documentos o vetor resultante da primeira etapa, chamado por ora de vetor base, os documentos são representados na forma de vetores numéricos com tamanho do vetor base, o valor de cada posição no documento codificado indica a presença de um *token* contido no vetor base.

Como exemplo os documentos V e W em sua forma codificada: $V_{encoded}$ e $W_{encoded}$ apresentada na Figura 8 tem o tamanho do vetor $TOKENS$ apresentado na Figura 7.

Figura 8 – Documentos V e W codificados

Fonte: Elaborada pelo autor

Os índices dos vetores codificados fazem referência as posições dos *tokens* no vetor $TOKENS$, assim o valor presente em cada índice dos vetores codificados indica o número de ocorrências de *tokens* contidos na respectiva posição do vetor $TOKENS$ nos documentos. A Figura 9 exemplifica este relacionamento.

Figura 9 – Relacionamento entre o vetor $TOKENS$, $V_{encoded}$ e $W_{encoded}$

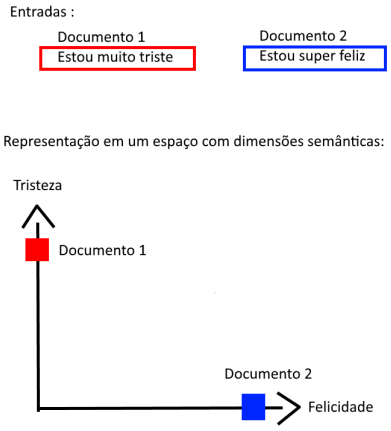
Fonte: Elaborada pelo autor

3.1.3 Word Embedding

Os modelos *Word Embedding* buscam um tipo de representação onde a posição no espaço de cada documento diz respeito de alguma forma ao seu conteúdo, neste tipo de representação são calculadas as posições dos documentos em um espaço de dimensões semânticas, esta técnica permite a visualização de documentos em espaços 2D e 3D fornecendo assim uma visualização gráfica do conteúdo dos mesmos, geralmente esta classe de representações utiliza processos de redução dimensional para representações dos dados.

Na Figura 10 dois documentos são representados através de *Word Embedding* em um espaço semântico de duas dimensões, onde cada dimensão diz respeito a um aspecto semântico do texto.

Figura 10 – Representado documentos em um espaço semântico



Fonte: Elaborada pelo autor

3.2 Processamento de dados textuais

Este capitulo é a fundamentação teórica dos algoritmos desenvolvidos neste trabalho para o processamento de dados textuais.

3.2.1 Similaridade de cossenos

A medida de similaridade de cossenos é utilizada para determinar a similaridade entre dois documentos, dado que estes estejam representados em forma vetorial, segue equação utilizada para o calculo:

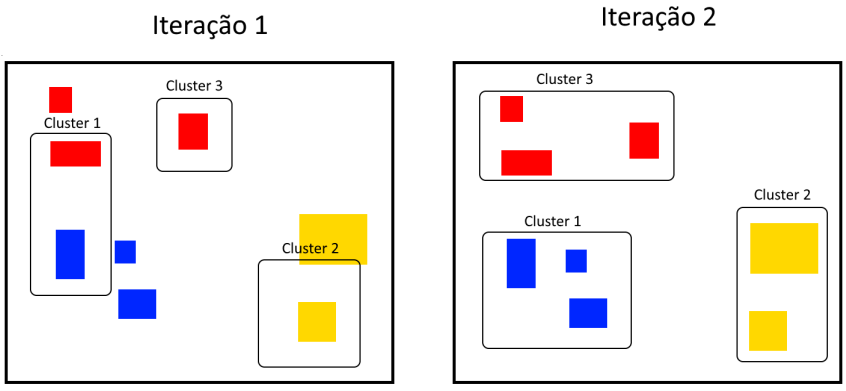
$$cos(\theta) = \frac{A.B}{|A||B|} \tag{3.1}$$

O calculo da similaridade de cossenos para documentos similares sintaticamente e codificados através de métodos de *Binary Encoding* retornará valores próximos a um, esta aplicação se relaciona com as teorias semânticas representacionalistas. Já para documentos codificados através de métodos de *Word Embedding*, o resultado da similaridade de cossenos se relacionará com significado das dimensões determinadas para o espaço semântico, retornando valores próximos a um para documentos similares semanticamente de acordo com o espaço, pode-se observar a semelhança da ideia de espaço semântico com a ideia de contexto, relacionando este tipo de aplicação e as teorias pragmáticas de significado.

3.2.2 Clustering

Algoritmos de *Clustering* particionam um conjunto de dados em grupos, este grupos são comumente conhecidos como *clusters*, o objetivo deste tipo de algoritmo é agrupar objetos similares. Existem duas principais classes de de técnicas para *Clustering*, a hierárquica e a não hierárquica, este trabalho enfoca somente na hierárquica.

Figura 11 – Iterações em um algoritmo de *Clustering* não hierárquico



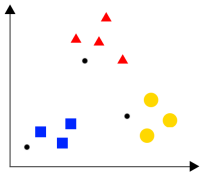
Fonte: Elaborada pelo autor

Algoritmos de clusterização não hierárquica geralmente começam com partições aleatoriamente selecionadas, e a cada iteração estas partições são melhoradas (Figura 11).

O algoritmo de clusterização implementado neste trabalho foi do tipo *K-means* baseado na implementação de Lloyds, tendo como referência o artigo (WILKIN; HUANG, 2007), segue um pseudo algoritmo desta implementação:

O primeiro passo é atribuir para cada *cluster* um ponto que será o seu centro inicial, a Figura 12 é uma representação do passo inicial do pseudo algoritmo, os pontos pretos representam o centro de cada *cluster* os demais representam dados.

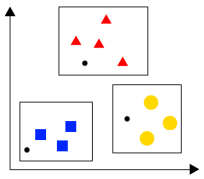
Figura 12 – Inciando algoritmo de *Clustering* com 3 *clusters*



Fonte: Elaborada pelo autor

O segundo passo é atribuir cada dado ao *cluster* mais próximo, ou seja com menor distância centro do *cluster* para o dado (Figura 13).

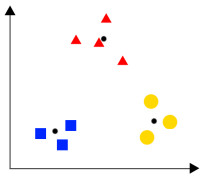
Figura 13 – Atribuindo dados ao *cluster* mais próximo



Fonte: Elaborada pelo autor

O terceiro passo é recomputar o ponto central de cada *cluster* com o valor da média dos dados atribuídos ao mesmo(Figura 14).

Figura 14 – Recalculando os pontos centrais de cada *cluster*



Fonte: Elaborada pelo autor

Este algoritmo pode ser aplicado para agrupar dados textuais, levando em conta aspectos semânticos e ou sintáticos.

3.2.3 Latent Semantic Indexing (LSI)

A técnica LSI projeta consultas(*queries*) e documentos em um espaço com dimensões semânticas latentes, termos que co ocorrem são projetados na mesmas dimensões e termos que não co ocorrem são projetados em dimensões diferentes.

Uma característica interessante desta técnica é que em um espaço semântico latente uma *query* e um documento pode ter alta similaridade de cosseno até se não compartilharem nenhum termo, pode-se evidenciar neste ponto a analise não somente das ocorrências dos termos, mas também do seu contexto trançando um relacionamento com as teorias semânticas pragmáticas.

Figura 15 – Matriz de termos em documentos

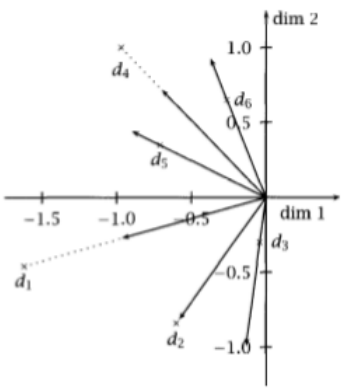
$A =$

	d_1	d_2	d_3	d_4	d_5	d_6
cosmonaut	1	0	1	0	0	0
astronaut	0	1	0	0	0	0
moon	1	1	0	0	0	0
car	1	0	0	1	1	0
truck	0	0	0	1	0	1

Fonte: MANNING; MANNING; SCHÜTZE, (1999)

O espaço semântico latente é um método de redução dimensional. Técnicas de redução de dimensionalidade são utilizadas também para tradução de objetos que existem num espaço altamente dimensional para um espaço com menos dimensões, geralmente em espaços com duas ou três dimensões para propósitos de visualização. A Figura 15 ilustra a representação de documentos em forma matricial, na Figura 16 tem-se documentos representados em um espaço semântico latente de duas dimensões .

Figura 16 – Representação de documentos em um espaço semântico latente



Fonte: Grossman ; Frieder (2012)

Este trabalho utiliza o método de Decomposição em Valores Singulares (*Singular Value Decomposition*, SVD) para análise de co ocorrência e redução dimensional de um espaço de documentos em um espaço semântico latente. O método de SVD projeta um espaço de n dimensões em um espaço de k dimensões onde $n > k$. Tem-se geralmente que n é o número de tipos de palavras em uma coleção e k é frequentemente escolhido entre 100 e 150.

Da Figura 17 a Figura 22 tem-se um exemplo de pseudoalgoritmo LSI utilizando SVD baseado na implementação de (Grossman ; Frieder, 2012), para uma coleção dos seguintes documentos:

a) $d1$: Shipment of gold damaged in a fire

b) $d2$: Delivery of silver arrived in a silver truck

c) $d3$: Shipment of gold arrived in a truck.

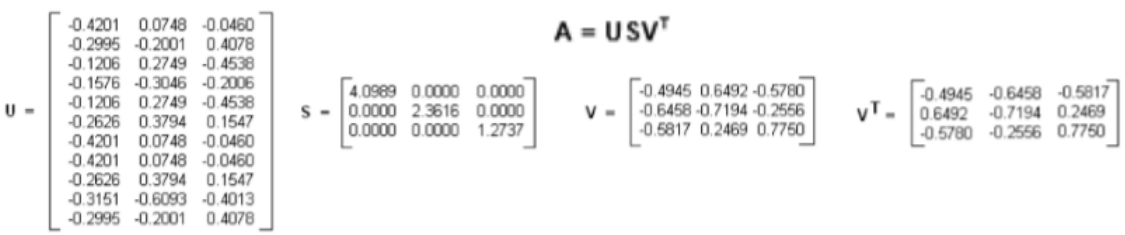
O primeiro passo para utilização do algoritmo é a representação dos documentos em forma matricial, esta pode ser obtida através do processo de *Binary Encoding* Figura 17.

Figura 17 – Primeiro passo representação dos documentos em forma matricial

Fonte: Grossman ; Frieder (2012)

O segundo passo é aplicação da decomposição em valores singulares na matriz obtida do passo um Figura 18.

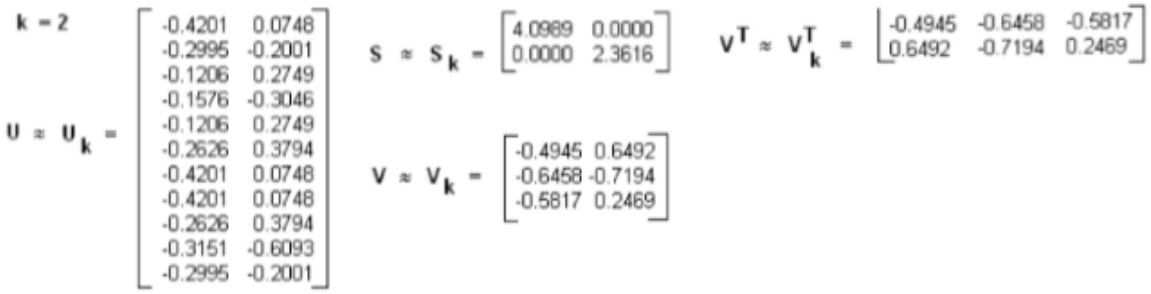
Figura 18 – Segundo passo decompor a matriz A utilizando SVD em $A = USV^t$



Fonte: Grossman ; Frieder (2012)

O terceiro passo é a obtenção da matriz de valores singulares a direita, e a aproximação desta utilizando o número de dimensões desejadas. Na Figura 19, é utilizada uma aproximação em duas dimensões.

Figura 19 – Terceiro passo realizar a diminuição dimensional, neste caso foi escolhida uma aproximação em duas dimensões



Fonte: Grossman ; Frieder (2012)

No quarto passo tem-se a representação dos documentos na forma de vetores em um espaço dimensionalmente reduzido Figura 20. Onde as coordenadas dos documentos são os valores da matriz de valores singulares a direita aproximados ao numero de dimensões desejadas.

Figura 20 – Quarto passo encontrar as coordenadas dos vetores que representam os documentos no espaço dimensionalmente reduzido

a) $d1 = (-0.4945, 0.6492)$
b) $d2 = (-0.6458, -0.7194)$
c) $d3 = (-0.5817, 0.2469)$

Em um quinto passo pode-se projetar *queries* em um espaço latente, possibilitando consultas em um espaço semântico latente, a formula utilizada para o processo é $q = q^t U_k S_k^{-1}$.

Figura 21 – Quinto passo projetar uma consulta (*query*) no espaço dimensionalmente reduzido $q = q^t U_k S_k^{-1}$.

Fonte: Grossman ; Frieder (2012)

No sexto passo através de similaridade de cossenos pode-se dizer quão similar uma *query* é de um documento Figura 22. Tem-se então um método para ordenação por similaridade semântica de documentos.

Figura 22 – Sexto passo ordenar os documentos de acordo com a similaridade dos cossenos.

$$sim(q, d) = \frac{q \bullet d}{|q||d|}$$

$$sim(q, d_1) = \frac{(-0.2140)(-0.4945) + (-0.1821)(0.6492)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\sqrt{(-0.4945)^2 + (0.6492)^2}} = -0.0541$$

$$sim(q, d_2) = \frac{(-0.2140)(-0.6458) + (-0.1821)(-0.7194)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\sqrt{(-0.6458)^2 + (-0.7194)^2}} = 0.9910$$

$$sim(q, d_3) = \frac{(-0.2140)(-0.5817) + (-0.1821)(0.2469)}{\sqrt{(-0.2140)^2 + (-0.1821)^2}\sqrt{(-0.5817)^2 + (0.2469)^2}} = 0.4478$$

Ranking documents in descending order

$$d_2 > d_3 > d_1$$

Fonte:Grossman ; Frieder (2012)

4 Implementação

A implementação da biblioteca foi realizada utilizando as linguagens JAVA e SCALA, as próximas seções tratam de aspectos da implementação.

4.1 Versionamento

Para o versionamento foi utilizado o sistema de controle de versão *open source*: Git. O projeto possui duas *branchs*: *develop* para controle do código em desenvolvimento e *master* contendo uma versão estável do código e pode ser acessado no repositório do github presente na URL: https://github.com/willer007/Simple-NLP.

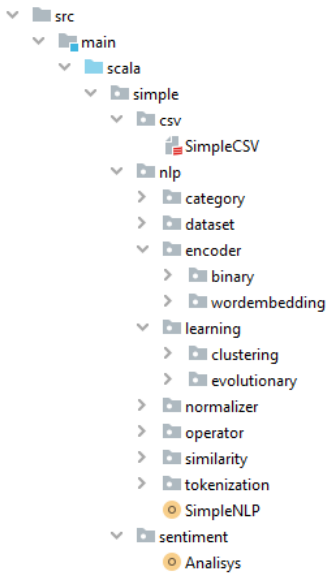
4.2 Automação de compilação

Para compilação do projeto foi utilizado o sistema de automação de compilação *open source gradle*.

4.3 Estrutura do projeto

Na Figura 23 esta a estrutura adotada para este projeto, cada seta presente na figura indica uma pasta.

Figura 23 – Estrutura de projeto



Fonte: Elaborada pelo autor.

4.4 Implementação dos Algoritmos

Neste capítulo são descritas as implementações dos algoritmos desenvolvidos neste trabalho bem como possibilidades de melhorias a serem desenvolvidas em trabalhos futuros.

4.4.1 Tokenização

Foram desenvolvidas três funções de tokenização, estas devem receber no mínimo um parâmetro $text$ do tipo $String$ que representa o texto a ser tokenizado e devolver um vetor do tipo $Array[String]$ contendo o texto na forma de um vetor de tokens.(Figura 24)

Figura 24 – Implementação de um algoritmo para Tokenização

```
package simple.nlp.tokenization

import java.text.Normalizer

object Tokenization {

  def tokenizeWithNormalizer (text:String, stopwords:Array[String]): Array[String] = {
    text.split( regex = "\\s")
      .map(s =>Normalizer.normalize(s.toLowerCase(), Normalizer.Form.NFD)
       .replaceAll( regex = "[^A-Za-z0-9\\s]", replacement = "")
       .trim())
      .filter(s => !s.isEmpty)
      .filter(s => !stopwords.contains(s))
  }

  def tokenizeByWhitespace (text:String): Array[String] = {
    text.split( regex = "\\s")
      .map(s => s.trim())
      .filter(s => !s.isEmpty)
  }


  def tokenizeCustom(text:String, regexTokenizerExpression:String): Array[String] = {
    text.split(regexTokenizerExpression)
      .filter(s => !s.isEmpty)
  }
}
```

Fonte: Elaborada pelo autor.

Para implementação futura sugere-se o desenvolvimento de bases nativas contendo palavras de parada (*stopwords*), bases de expressões regulares que contemplem as diferenças lexicais de cada linguagem e utilização destas para melhorias na implementação dos algoritmos presentes.

4.4.2 Binary Encoding

A primeira etapa para utilização do algoritmo de *Binary Encoding* desenvolvido neste trabalho é a chamada do método $loadTokens(setTokens : Set[String])$ este método requer um parâmetro do tipo $Set[String]$ que deverá conter todos os *tokens* presentes na base a ser

codificada, um Set é uma estrutura de dados que não aceita repetição assim este algoritmo garante que o Set carregado não conterá *tokens* duplicados.

Após executada a primeira etapa pode-se realizar a codificação de texto tokenizado para o formato de um vetor de inteiros $Array[Int]$ pelo método $encode(arrayToken : Array[String])$.

O método $encode(arrayToken : Array[String])$ retorna um vetor de inteiros do tamanho de $setTokens$ considerando o número de ocorrências de cada *token* no texto, para realização da codificação sem consideração do número de ocorrências dos *tokens* nos documentos podemos utilizar o método $encodeWithoutCount(arrayToken : Array[String])$ que considera apenas a ocorrência ou não do termo no texto.

Figura 25 – Implementação de um algoritmo para *Binary Encoding*

```
object EncoderBinary {
  private var tokenEncoder: Map[String, Int] = Map.empty
  private var tokenDecoder: Map[Int, String] = Map.empty
  private var numberOfTokens = 0

  def loadTokens(setTokens: Set[String]): Unit = {
    numberOfTokens = 0
    for (t <- setTokens) {
      tokenEncoder += (t -> numberOfTokens)
      tokenDecoder += (numberOfTokens -> t)
      numberOfTokens += 1
    }
  }

  def encode(arrayToken: Array[String]): Array[Int] = {
    var tokensEncoded = Array.ofDim[Int](numberOfTokens)
    arrayToken.foreach(token => {
      val tokenPosition = tokenEncoder(token)
      var tokenOccurrence = (tokensEncoded(tokenPosition))
      tokenOccurrence = (tokenOccurrence + 1)
      tokensEncoded(tokenPosition) = tokenOccurrence
    })
    tokensEncoded
  }

  def encodeWithoutCount(arrayToken: Array[String]): Array[Int] = {
    var tokensEncoded = Array.ofDim[Int](numberOfTokens)
    arrayToken.foreach(token => {
      tokensEncoded(tokenEncoder(token)) = 1
    })
    tokensEncoded
  }

  def decode(arrayToken: Array[Int]): Array[String] = {
    var stringDecoded: Array[String] = Array.empty
    for (tokenPosition <- arrayToken.indices) {
      if (arrayToken(tokenPosition) >= 1) { stringDecoded = tokenDecoder(tokenPosition) +: stringDecoded }
    }
    stringDecoded
  }
}
```


Fonte: Elaborada pelo autor.

Para a realizar a operação inversa, ou seja a decodificação de um vetor de inteiros em um vetor de *tokens* podemos utilizar o método $decode(arrayToken : Array[Int])$. (Figura 25)

4.4.3 Similaridade de Cossenos

Para o calculo da similaridade pode-se utilizar a função $similarityCossine(v1 : Array[Double], v2 : Array[Double])$ que devolve o coeficiente de similaridade entre dois vetores em um valor tipo $Double$.(Figura 26)

Figura 26 – Implementação de um algoritmo para Similaridade de cossenos

```
package simple.nlp.similarity

import simple.nlp.operator.OperatorArray


object SimilarityCossine {

  def similarityCossine(v1:Array[Double], v2:Array[Double]): Double =
    (v1 zip v2).map(t => t._1*t._2).sum / (OperatorArray.arrayModulus(v1) * OperatorArray.arrayModulus(v2))


}
```

Fonte: Elaborada pelo autor.

4.4.4 Clustering

A implementação de *Clustering* realizada neste trabalho foi baseada no algoritmo não hierárquico de Lloyds e permite a personalização do número de *clusters* ($numOfClusters$) bem como o número de dimensões de cada *cluster* ou seja o tamanho dos vetores de entrada($numOfDimension$), após definidos estes parâmetros pode-se inicializar o algoritmo através do método $optimize(inputs : Array[Array[Float]], iteration : int)$, este método recebe como parâmetros de entrada um vetor de vetores do tipo $Array[Array[Float]]$ e o número de iterações a ser realizadas pelo algoritmo e devolve um vetor de objetos do tipo $Cluster$.(Figura 27)

Figura 27 – Clustering

```
package nlp.learning.clustering

import simple.nlp.learning.clustering.LearningLloyds

object LearningLloydsTest {

  def main(args: Array[String]): Unit = {

    LearningLloyds.initModel( numOfClusters = 10,  numOfDimension = 10)

    var inputs: Array[Array[Float]] = Array.empty
    for (num <- 1 to 10) {
      inputs = inputs :+ Array.fill[Float](10)(num.toFloat)
    }


    var test = LearningLloyds.optimize(inputs, iterations = 100)

  }
}
```

Fonte: Elaborada pelo autor.

Os objetos do tipo $Cluster$ possuem dois atributos $centroid$ um vetor do tipo $Array[Float]$, que representa o centro do *cluster*, e $data$ um vetor de vetores do tipo $ArrayBuffer[Array[Foat]]$, que contém os dados atribuídos a este *cluster*.(Figura 28)

Figura 28 – Cluster

```
package simple.nlp.learning.clustering

import scala.collection.mutable.ArrayBuffer
import scala.util.Random


case class Cluster(centroid: Array[Float], data: ArrayBuffer[Array[Float]])

object Cluster {
  def addData(cluster: Cluster, data: Array[Float]): Unit = cluster.data += data
  def setCentroid(cluster: Cluster, centroid: Array[Float]) = Cluster(centroid, cluster.data)
  def createCluster(size:Int): Cluster = Cluster(Array.fill(size) {Random.nextFloat()}, ArrayBuffer.empty)
  def createCluster(centroid: Array[Float]):Cluster = Cluster(centroid,ArrayBuffer.empty)

}
```

Fonte: Elaborada pelo autor.

4.4.5 Latent Semantic Indexing (LSI)

A primeira etapa para utilização do algoritmo LSI implementado neste trabalho é a chamada do método $encode(inputs : Array[Vector], numOfDimensions : Int)$ este método realiza a modelagem das entradas $inputs$ para o modelo *Word Embending* de dimensões $numOfDimensions$ após codificados os dados, pode-se codificar as *querys* a ser utilizadas, com o método $encodeQuery(query : Array[Double])$.(Figura 29

Figura 29 – Implementação do algoritmo LSI

```
object EncoderLSITest {

  def main(args: Array[String]): Unit = {

    var inputs:Array[Vector] = Array.empty


    inputs = inputs :+ Vectors.dense( firstValue = 1, otherValues = 0,1,0,1,1,1,1,1,0,0)
    inputs = inputs :+ Vectors.dense( firstValue = 1, otherValues = 1,0,1,0,0,1,1,0,2,1)
    inputs = inputs :+ Vectors.dense( firstValue = 1, otherValues = 1,0,0,0,1,1,1,1,0,1)

    var test = EncoderLSI.encode(inputs, numOfDimensions = 2)

    var query:Array[Double] = Array(0,0,0,0,0,1,0,0,0,1,1)
    var testQuery = EncoderLSI.encodeQuery(query)


  }

}
```

Fonte: Elaborada pelo autor.

5 Aplicação

Na Figura 30 estão aplicados a análise de *tweets* uma forma de cada algoritmo desenvolvido neste trabalho exceto do algoritmo de clusterização, o objetivo dessa aplicação é verificar o relacionamento entre os documentos contidos em um dataset de *tweets* e as expressões "sad"e "happy".

Figura 30 – Aplicação do algoritmo LSI

```
//NORMALIZAÇÃO E CARREGAMENTO DO DATASET
var dataset = SimpleCSV.readCSV( path = "C:\\Users\\Willer\\Desktop\\tcc_dataset\\tcc_dataset_tweets_100k.csv")
  .replaceAll( regex = "[,.!?\"]", replacement = "")
  .replaceAll( regex = "@\\w+", replacement = "")
  .replaceAll( regex = "http.+\\s", replacement = "")
  .toLowerCase()

//SEPARAÇÃO DO DATASET EM MULTIPLAS LINHAS
var datasetRows = dataset.split( regex = "\r\n").slice(0,1000)

//CARREGAMENTO DE TODOS OS TOKENS PERTENCENTES AO DATASET
var allTokens: Set[String] = SimpleNLP.Tokenizator.tokenizeByWhitespace(datasetRows.reduce((a,b) => a + " " + b)).toSet
SimpleNLP.EncoderBinary.loadTokens(allTokens)

//TOKENIZACAO
var datasetTokenized = datasetRows.map(row => SimpleNLP.Tokenizator.tokenizeByWhitespace(row))
//ENCODING
var datasetEncoded = datasetTokenized.map(row => SimpleNLP.EncoderBinary.encodeWithoutCount(row))
//EXPRESSAO FELIZ
var happyExpression =  SimpleNLP.EncoderBinary.encodeWithoutCount(Array("happy"))
//EXPRESSAO TRISTE
var sadExpression =  SimpleNLP.EncoderBinary.encodeWithoutCount(Array("sad"))
//ENCODE SEGUNDO MODELO LSI
var encodedDataset   =datasetRows zip SimpleNLP.EncoderLSI.encode(datasetEncoded, numOfDimensions = 30)
var happyLSI = SimpleNLP.EncoderLSI.encodeQuery(happyExpression)
var sadLSI = SimpleNLP.EncoderLSI.encodeQuery(sadExpression)
//EXPRESSAO FELIZ MODELO LSI
var happy = encodedDataset.
  sortBy(data => SimpleNLP.SimilarityCossine.similarityCossine(data._2,happyLSI))
var happyText:String = happy.map(d => d._1).reduce((a,b) =>a + "\n" +b)
SimpleCSV.writeCSV( path = "C:\\Users\\Willer\\Desktop\\tcc_dataset\\happy.txt",happyText)
//EXPRESSAO TRISTE MODELO LSI
var sad = encodedDataset.
  sortBy(data => SimpleNLP.SimilarityCossine.similarityCossine(data._2,sadLSI))
var sadText:String = sad.map(d => d._1).reduce((a,b) =>a + "\n" +b)
SimpleCSV.writeCSV( path = "C:\\Users\\Willer\\Desktop\\tcc_dataset\\sad.txt",sadText)
```

Fonte: Elaborada pelo autor.

Para análise dos *tweets* realizada na figura acima a primeira etapa foi o carregamento do *dataset* "tcc_dataset_tweets_100k.csv"contendo cerca de 100 mil tweets no formato de planilha csv, com cada linha da planilha representando um *tweet*. Figura 31.

Figura 31 – Aplicação do Algoritmo LSI primeira etapa

```
//NORMALIZAÇÃO E CARREGAMENTO DO DATASET
var dataset = SimpleCSV.readCSV( path = "C:\\Users\\Willer\\Desktop\\tcc_dataset\\tcc_dataset_tweets_100k.csv")
  .replaceAll( regex = "[,.!?\"]", replacement = "")
  .replaceAll( regex = "@\\w+", replacement = "")
  .replaceAll( regex = "http.+\\s", replacement = "")
  .toLowerCase()

//SEPARAÇÃO DO DATASET EM MULTIPLAS LINHAS
var datasetRows = dataset.split( regex = "\r\n").slice(0,1000)
```

Fonte: Elaborada pelo autor.

A segunda etapa foi a extração e carregamento de todos os *tokens* presentes nos *tweets* no algoritmo de *Binary Encoding* ver Figura 32.

Figura 32 – Aplicação do Algoritmo LSI segunda etapa

```
//CARREGAMENTO DE TODOS OS TOKENS PERTENCENTES AO DATASET
var allTokens: Set[String] = SimpleNLP.Tokenizator.tokenizeByWhitespace(datasetRows.reduce((a,b) => a + " " + b)).toSet
SimpleNLP.EncoderBinary.LoadTokens(allTokens)
```

Fonte: Elaborada pelo autor.

Na terceira etapa foram tokenizadas todas as linhas do *dataset*. Figura 33.

Figura 33 – Aplicação do Algoritmo LSI terceira etapa

```
//TOKENIZACAO
var datasetTokenized = datasetRows.map(row => SimpleNLP.Tokenizator.tokenizeByWhitespace(row))
```

Fonte: Elaborada pelo autor.

Na quarta etapa foram codificadas segundo o algoritmo de *Binary Encoding*, todas as linhas do *dataset*. Figura 34

Figura 34 – Aplicação do Algoritmo LSI quarta etapa

```
//ENCODING
var datasetEncoded = datasetTokenized.map(row => SimpleNLP.EncoderBinary.encodeWithoutCount(row))
```

Fonte: Elaborada pelo autor.

Já na quinta etapa foram codificadas utilizando *Binary Encoding* os termos "happy"e "sad". Figura 35

Figura 35 – Aplicação do Algoritmo LSI quinta etapa

```
//EXPRESSAO FELIZ
var happyExpression =  SimpleNLP.EncoderBinary.encodeWithoutCount(Array("happy"))

//EXPRESSAO TRISTE
var sadExpression =  SimpleNLP.EncoderBinary.encodeWithoutCount(Array("sad"))
```

Fonte: Elaborada pelo autor.

Na sexta e última etapa o algoritmo LSI foi aplicado ao *dataset* codificado no modelo *Binary Encoding*. Os documentos foram representados em um espaço *Word Embbeding* de 30 dimensões (Figura 36). Foram projetados neste espaço os termos "happy"e "sad"e utilizados respectivamente para a geração de dois documentos "happy.txt"e "sad.txt", que contem *tweets* ordenados em ordem de similaridade de cossenos crescente com a expressão em seu nome.

Assim sendo o primeiro documento de cada arquivo, contém em sua primeira linha o documento menos similar ao seu nome e em sua última linha o documento mais similar.

Figura 36 – Aplicação do algoritmo LSI sexta etapa

```
//ENCODE SEGUNDO MODELO LSI
var encodedDataset   =datasetRows zip SimpleNLP.EncoderLSI.encode(datasetEncoded, numOfDimensions = 30)
var happyLSI = SimpleNLP.EncoderLSI.encodeQuery(happyExpression)
var sadLSI = SimpleNLP.EncoderLSI.encodeQuery(sadExpression)

//EXPRESSAO FELIZ MODELO LSI
var happy = encodedDataset.
  sortBy(data => SimpleNLP.SimilarityCossine.similarityCossine(data._2,happyLSI))
var happyText:String = happy.map(d => d._1).reduce((a,b) =>a + "\n" +b)
SimpleCSV.writeCSV( path = "C:\\Users\\Willer\\Desktop\\tcc_dataset\\happy.txt",happyText)

//EXPRESSAO TRISTE MODELO LSI
var sad = encodedDataset.
  sortBy(data => SimpleNLP.SimilarityCossine.similarityCossine(data._2,sadLSI))
var sadText:String = sad.map(d => d._1).reduce((a,b) =>a + "\n" +b)
SimpleCSV.writeCSV( path = "C:\\Users\\Willer\\Desktop\\tcc_dataset\\sad.txt",sadText)
```

Fonte: Elaborada pelo autor.

Da analise dos documentos processados o documento menos similar a expressão happy foi: - i think i need to find better anti-depressants i think this paxil/wellbutrin combo is losing its efficacy", enquanto que o documento mais similar a expressão sad foi : "that's sad", observa-se uma narrativa depressiva como a menos similar a happy, enquanto que as narrativas mais similares a "sad"não possuem um relacionamento explicito com a depressão, concluindo é possível que haja um relacionamento entre a ausência de termos relacionados a felicidade e a depressão.

A técnica de *Clustering* foi aplicada aos dados no modelo de *Word Embbeding* obtidos na figura (Figura 36), contudo os resultados foram inconclusivos(Figura 37), os agrupamentos foram inconsistentes

Figura 37 – Aplicação do algoritmo de Clustering

```
//ENCODE SEGUNDO MODELO LSI
var encodedDataset   = datasetRows zip SimpleNLP.EncoderLSI.encode(datasetEncoded, numOfDimensions = 30)
var docsPositions = encodedDataset.map(d => d._2)

//CLUSTERING
SimpleNLP.LearningLloyds.initModel( numOfClusters = 10, numOfDimension = 30)
var dataClusterized = SimpleNLP.LearningLloyds.optimize(docsPositions, iterations = 50)
```

Fonte: Elaborada pelo autor.

6 Conclusão

A linguagem SCALA possibilita o desenvolvimento de função matemáticas de forma simples e clara uma vez que utiliza o paradigma funcional da programação (CHIUSANO; BJARNASON, 2014), que tenta aplicar conceitos da matemática ao desenvolvimento de software, esta propriedade facilitou o desenvolvimento dos algoritmos apresentados neste trabalho, principalmente dos algoritmos de processamento de dados textuais.

Foi observado o relacionamento das teorias filosóficas semânticas representacionalistas da linguagem com as formas de representação de textual para processamento de linguagem natural mais diretas como texto tokenizado e *Binary Encoding*, estas formas de representação parecem ser mais apropriadas a analise sintática, pois são representação crua dos termos envolvidos no texto de forma vetorial.

Já as representações com técnicas de *Word Embbeding* levam em conta aspectos específicos do texto para elaboração do espaço onde serão apresentados os documentos, temos que nestes modelos cada dimensão do espaço diz respeito a um aspecto textual, o conjunto dos aspectos analisados pode ser dito contexto, o contexto para a técnica LSI aparece na forma da co-ocorrência de termos em documentos, podemos traçar um relacionamento deste tipo de analise com as teorias pragmáticas da linguagem, este tipo de representação é útil a analise semântica de texto, pois entende como representativo os aspectos delimitados no espaço de *Word Embbeding* e não somente a sintaxe.

Para o processamento de linguagem natural técnicas de *Clustering* são úteis para agrupamento de dados textuais, em representações de dados textuais que utilizam *Binary Encoding* a aplicação de *Clustering* é recomendada quando se quer levar em conta apenas o aspecto sintático da linguagem durante o agrupamento, enquanto que representações utilizando *Word Embbending* são mais adequadas quando se deseja agrupar texto de acordo com aspectos semânticos.

O algoritmo LSI permitem representar dados textuais em um espaço de dimensões semânticas, consultas (*queries*) projetadas neste espaço possibilitam verificar quão semanticamente próximos estão documentos, sendo assim uma ferramente extremamente útil a analise semântica de documentos.

Referências

CHIUSANO, P.; BJARNASON, R. *Functional programming in Scala*. [S.l.]: Manning Publications Co., 2014.

LEITE, J. E. R. Fundamentos da linguística. *Lingua Portuguesa e Libras: teorias e práticas. Joao Pessoa. Ed Universitaria da UFPB*, v. 1, p. 171–232, 2010.

MANNING, C. D.; MANNING, C. D.; SCHÜTZE, H. *Foundations of statistical natural language processing*. [S.l.]: MIT press, 1999.

NIGRO, R. A virada linguístico-pragmática e o pós-positivismo. *Revista Direito, Estado e Sociedade*, n. 34, 2009.

PINHEIRO, V. *SIM: Um Modelo Semântico Inferencialista para Expressão e Raciocínio em Sistemas de Linguagem Natural*. Tese (Doutorado) — Phd Thesis, Universidade Federal do Ceará, 2010.

PRIBERAM, 2019. In: DICIONáRIO da língua portuguesa. Priberam Informática, 2019. Disponível em: <https://dicionario.priberam.org/>.

TEIXEIRA, E. N.; MARTINS, H. F. Curso de lingüística geral: Reação e adesão à perspectiva representacionista. *RevEL. Edição especial*, n. 2, 2008.

WEBSTER, J. J.; KIT, C. Tokenization as the initial phase in nlp. In: *COLING 1992 Volume 4: The 15th International Conference on Computational Linguistics*. [S.l.: s.n.], 1992.

WILKIN, G. A.; HUANG, X. K-means clustering algorithms: implementation and comparison. In: IEEE. *Second International Multi-Symposiums on Computer and Computational Sciences (IMSCCS 2007)*. [S.l.], 2007. p. 133–136.